



Automating Data Layout Conversion in a Large Cosmological Simulation Code

Michele Martone, Luigi Iapichino, Nicolay Hammer, Julia Lawall

► To cite this version:

Michele Martone, Luigi Iapichino, Nicolay Hammer, Julia Lawall. Automating Data Layout Conversion in a Large Cosmological Simulation Code. CoSaS 2018 - International Symposium on Computational Science at Scale, Sep 2018, Erlangen, Germany. hal-01890314

HAL Id: hal-01890314

<https://hal.inria.fr/hal-01890314>

Submitted on 8 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Automating Data Layout Conversion in a Large Cosmological Simulation Code

Michele MARTONE, Luigi IAPICHINO, Nicolay HAMMER, Julia LAWALL*

Leibniz Supercomputing Centre, Garching bei München, Germany

*INRIA/Laboratoire Informatique Paris-6, Paris, France



Intro: The Gadget cosmological code

- Large-scale structure formation (galaxies and clusters)
- Publicly available [1], cosmological TreePM N-body + SPH code
- MPI/OpenMP-parallel; up to O(100k) Xeon cores (SuperMUC@LRZ)
- Several teams and several versions (>200 kLoC each)

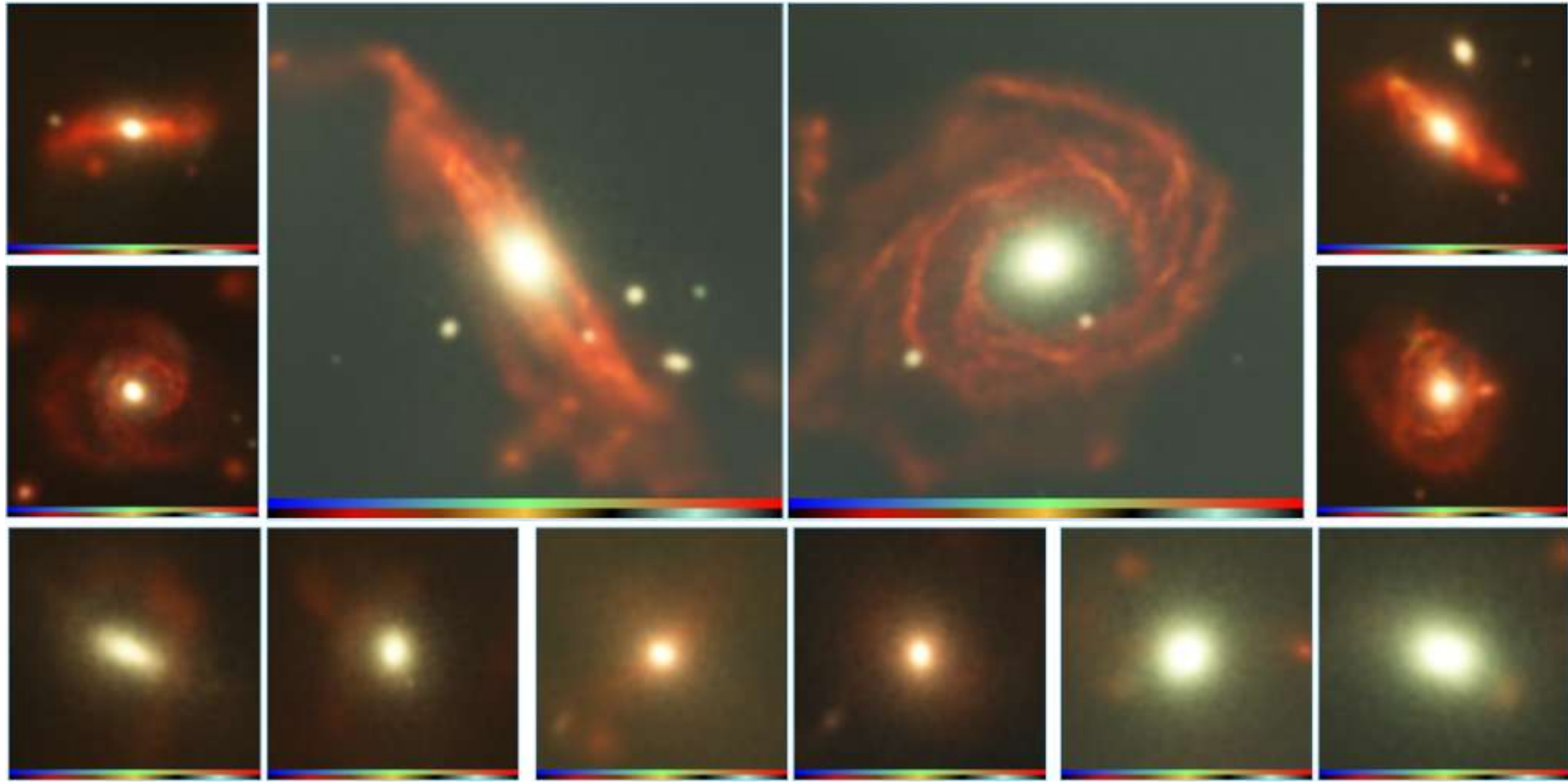


Figure 1: Galaxies simulated with Gadget – courtesy Magneticum Project [2] / <http://magneticum.org>

Optimization study of Gkernel

- Gkernel is:
 - isolated representative Gadget code kernel (“halo finder”)
 - stand-alone application, avoids simulation overhead
- Node-level optimization study [3] in the frame of the IPCC [4]
- Target computing systems:
 - Knights Corner (KNC), Ivy Bridge (IVB)
 - Haswell (HSW), Broadwell (BDW), Knights Landing (KNL)
- Main changes:
 - Data layout optimization
 - Better threading parallelism

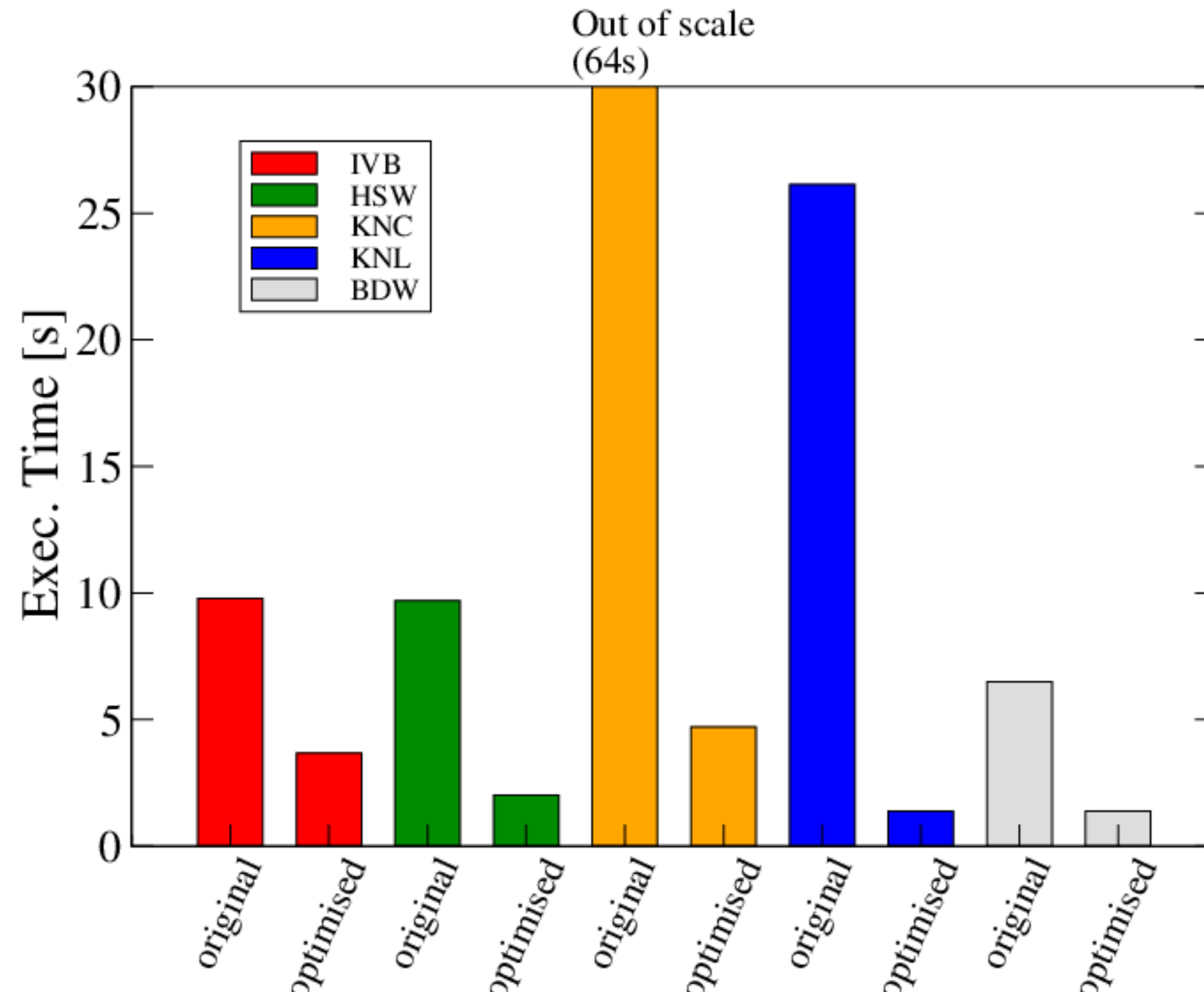


Figure 2: Tests on one-socket Xeon systems; 240 threads (4 thr./core) for KNC; 128 threads (2 thr./core) for KNL. Performance improvement: up to 19x faster on KNL; 13.6x on KNC, ca. for 2-5x on Xeon.

Data layout optimization

From Array of Structures to Structure of Arrays (AoS \Rightarrow SoA)

- Sample code changes in Fig. 3 and Fig. 4
- Leads to better memory access
- Eases compiler *auto-vectorization*
- SoA-specific performance benefit (over OpenMP improvement):
 - +13% on IVB
 - +48% on KNC

```
1 // Before:
2 // one structure per particle
3 struct particle {
4     double Mass, Hsml, ...;
5     ...
6 };
7
8 ...
9 // Array of Structures
10 struct particle *P;
```

```
1 // After:
2 // One array for each quantity
3 struct particle_soa {
4     double *Mass, *Hsml, ...;
5     ...
6 };
7
8 ...
9 // Structure of Arrays
10 struct particle_soa_t P_SoA;
```

Figure 3: A “Structure of Arrays” struct definition introduction example. Only a dozen arrays have been introduced this way in Gkernel. Gadget would need over a hundred.

```
1 // may not vectorize
2 ...P[i].Mass + P[i]...
```

```
1 // vectorizes better
2 ...P_SoA.Mass[i] + P_SoA...
```

Figure 4: AoS to SoA transition sample. Impacts a few hundreds statements in Gkernel. Would impact many thousands in full Gadget. Notice that i could be any expression.

Perspective for Gadget

- Obtained and validated performance guidelines for Gadget
- Limitations of [3] study:
 - single kernel only, no MPI
 - relying on $AoS \Rightarrow SoA \Rightarrow AoS$ at each kernel call
- Left open questions:
 - what to do with members like Pos [3] or DV [3] [3] ?
 - ...and *anonymous unions* ?
 - ...and other kernels/quantities?
- How to backport $\gg 200$ kLoC to Gadget ?

Discussion: Possible AoS \Rightarrow SoA breakdown

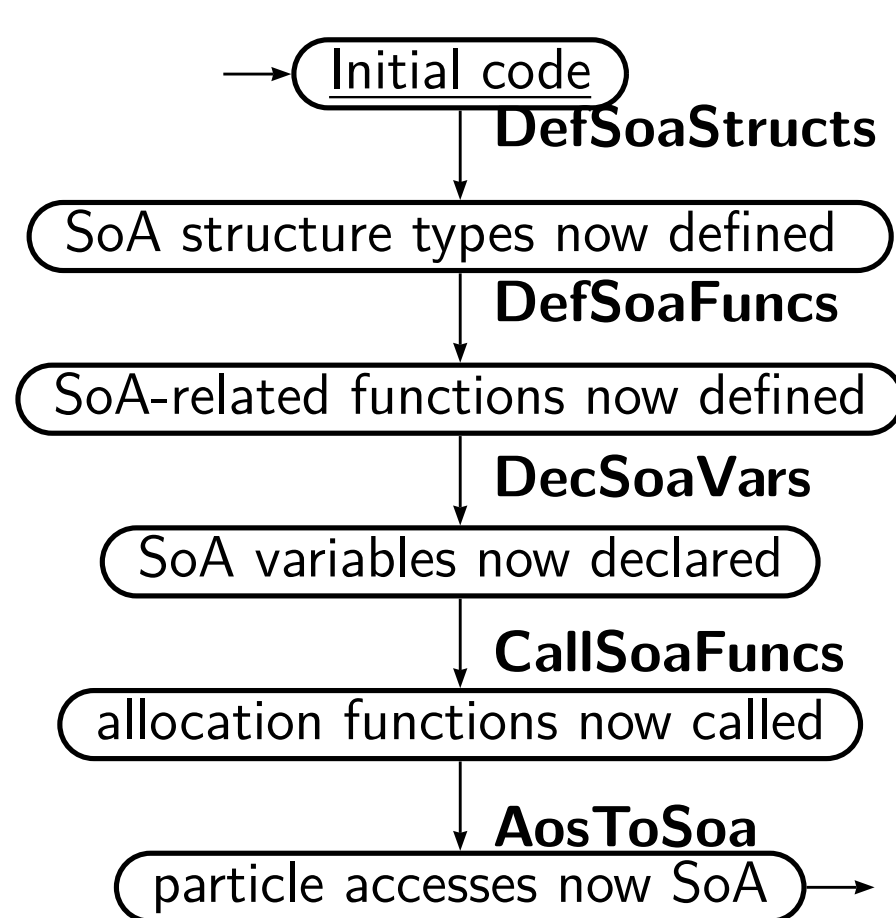


Figure 5: Transformation steps, ordered so that their introduction does not break code. The Gkernel study has performed this on a subset of Gadget ([3]).

- DefSoaStructs: define struct type with arrays instead of scalars
 - each member might depend on `#ifdef`...
- DefSoaFuncs: glue functions for SoA
 - like `-allocate`, `-free`, `-convert`, etc.
 - repetitive, time consuming

```
1 void aos_alloc(void)
2 {
3     P = (struct particle*) calloc(
4         All.MaxPart, struct particle)
5     ;
6     // all particle data allocated
7 }
```

```
1 void soa_alloc(void)
2 {
3     P_SoA.Hsml = (double*) calloc(
4         All.MaxPart, double);
5     P_SoA.Mass = (double*) calloc(
6         All.MaxPart, double);
7     ...
8 }
```

Figure 6: DefSoaFuncs. Transition to SoA in Gadget would require one allocation per quantity. Manual introduction of this and further glue functions (e.g. $AoS \Leftrightarrow SoA$ copy functions) is error-prone and repetitive. Manual SoA accesses introduction (Fig. 4) can be worse.

- DecSoaVars: declare SoA structure variables
- CallSoaFuncs: call glue functions
- AosToSoa: replace relevant AoS accesses with SoA accesses
 - thousands of lines impacted (error prone and repetitive)
 - regular expressions help, but still error-prone and sloppy

A more desirable approach

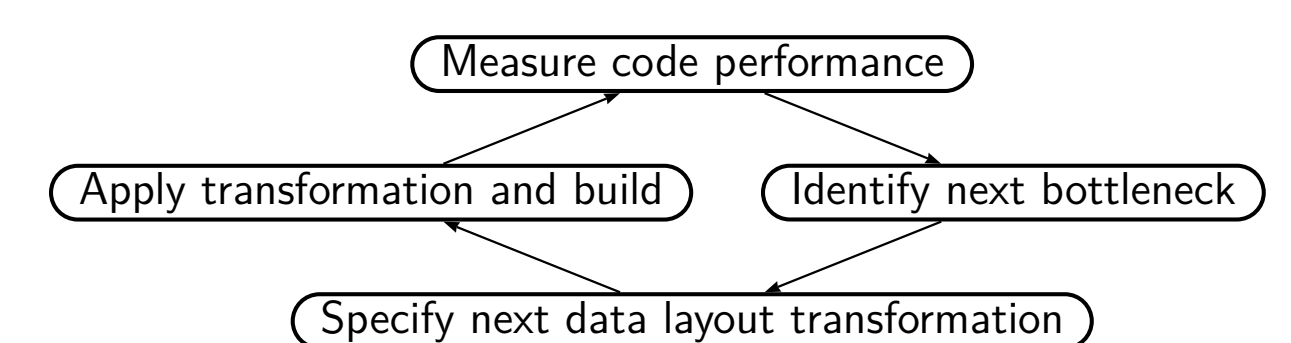


Figure 7: Tool-assisted automatic data layout transformation would greatly accelerate cycles of development and empirical performance optimization.

Fine-grained AoS \Rightarrow SoA transition mechanism needed!

- transitory patch: generating $AoS \Leftrightarrow SoA$ converter functions
- exclusion/inclusion of structure members based on
 - usage / occurrence in files
 - specific type
 - identifier whitelist

Solution: Semantic patching

Coccinelle [5]: “...a program matching and transformation engine ... for specifying desired matches and transformations in C code”

- specification language inspired from UNIX patch (SmPL)
- SmPL can express steps in Fig. 5, enables Fig. 7 work style
- originally for collateral evolutions in the Linux kernel
- arbitrary C language transformations possible, model checking

```
1 @@
2 identifier id,I;
3 type T;
4 @@
5 struct id { ...
6     T I;
7     T *I;
8     ...
9 };
```

```
1 @@
2 expression E;
3 identifier AoS;
4 identifier J;
5 fresh identifier SoA=AoS##"_SoA";
6 @@
7 - AoS[E].J
8 + SoA.J[E]
9
```

Figure 8: Sample semantic patch for step DefSoaStructs, cf. Fig. 3. Lines 2-3 specify semantic elements of the C language. Lines 5 and 9 specify matching context, line 6 what to add, line 7 what to remove.

Figure 9: Semantic patch example for AOsToSoa, cf. Fig. 4. This patch requires no context apart from the elements to be replaced. Notice also introduction of a new identifier at line 5, used at line 8.

```
1 @str_flds@
2 identifier id,I;
3 type T;
4 @@
5 struct id{ ...
6     T *I;
7     ...
8 };
9
10 @def_alloc_sig@
11 identifier str_flds.id;
12 @@
13 struct id { ... };
14 +void alloc(struct id*p,int n)
15 +{}
```

```
16 @def_alloc_body@
17 type str_flds.T;
18 identifier str_flds.I, str_flds.id;
19 @@
20 void alloc(struct id*p, int n){
21     ++p->I = calloc(n, sizeof(T));
22 }
23
24 @call_alloc@
25 identifier V, str_flds.id;
26 @@
27 struct id V;
28 + alloc(&V, 5);
29 +}
```

Figure 10: Semantic patch composed by four rules. This is similar to how DefSoaFuncs and CallSoaFuncs rules might look like. The first rule (line 1) matches a structure, the second (line 10) introduces an empty function, the third (line 16) populates it, the fourth (line 24) calls it. Notice inheritance of elements across rules, e.g. on line 11.

Coccinelle advantages

- over other transformation engines (e.g. DMS, Rose Compiler):
 - straight-forward transformation language SmPL
 - fully free software-licensed (it serves the Linux kernel)
- over certain syntax-preserving C++ techniques (e.g. Intel SDLT)
 - code clarity (no operators and temporary objects involved)
 - preserves usability of `-O0`

Results summary

- all scalar particle variables now SoA
- enables low effort extension to:
 - replay on several communities’ Gadget forks
 - arbitrary mixes of AoS/SoA
- patch: $\approx 10K$ +/- diff lines (with context: $\approx 20K$)
- exploring Coccinelle usage for HPC

Possible future work

- new performance studies possible
- rules for OpenMP optimizations backport [3]
- MPI-specific glue code

References

- V. Springel, “The cosmological simulation code GADGET-2”, *MNRAS*, vol. 364, pp. 1105–1134, 2005.
- “Magneticum Home page (large-scale project based on Gadget)”, <http://www.magneticum.org>.
- F. Baruffa, L. Iapichino, N. J. Hammer, and V. Karakasis, “Performance optimisation of smoothed particle hydrodynamics algorithms for multi-/many-core architectures”, in *2017 International Conference on High Performance Computing Simulation (HPCS)*, pp. 381–388, 2017.
- “Intel Parallel Computing Center: LRZ webpage”, <https://www.lrz.de/services/compute/labs/astrolab/ipcc>.
- J. Lawall and G. Muller, “Coccinelle: 10 years of automated evolution in the Linux kernel”, in *2018 USENIX Annual Technical Conference, USENIX ATC*, pp. 601–614, 2018.